

PATENT ABSTRACTS OF JAPAN

(11)Publication number : 10-207929

(43)Date of publication of application : 07.08.1998

(51)Int.Cl.

G06F 17/50
G06F 9/06

(21)Application number : 09-010821

(71)Applicant : NEC CORP

GIJUTSU KENKYU KUMIAI
SHINJOHO SHIYORI KAIHATSU
KIKO

(22)Date of filing : 24.01.1997

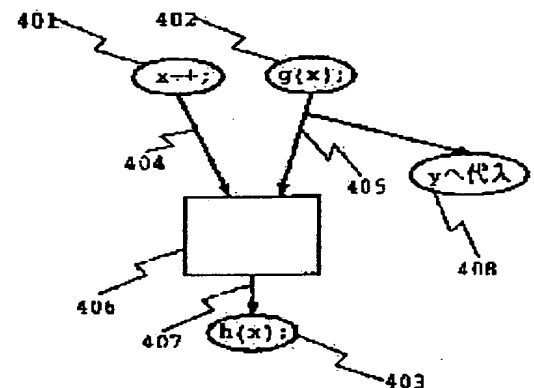
(72)Inventor : YAMAUCHI SO

(54) ARITHMETIC DEVICE SYNTHESIS METHOD

(57)Abstract:

PROBLEM TO BE SOLVED: To take out high parallelism provided in the concept of a data flow while using a procedure type language in method for directly mapping a program corresponding to a data/control flow to a conductor integrated circuit sand executing it.

SOLUTION: A scope stack for recursively analyzing the reference substitution relation of a variable in the large and a function and holding the substitution relation of the variable in the large and function calling and an arithmetic operation is generated. A circuit for finding the arithmetic operation and the function calling substituted for the variable in the large referred to by the function by tracing back the scope stack at the time of mapping the function calling to the semiconductor integrated circuit and activating the function calling when the two conditions that the argument of the function are arranged and the arithmetic operation and the function calling substituted for the variable in the large referred to by the function are ended corresponding to the scope stack are established is generated.



LEGAL STATUS

[Date of request for examination]

24.01.1997

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

2990084

[Date of registration]

08.10.1999

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's

[decision of rejection]

[Date of extinction of right]

Copyright (C); 1998,2003 Japan Patent Office

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開平10-207929

(43) 公開日 平成10年(1998)8月7日

(51) Int.Cl. ⁸	識別記号	F I
G 0 6 F 17/50		G 0 6 F 15/60
9/06	5 3 0	9/06
		6 5 4 M
		5 3 0 V

審査請求 有 請求項の数 8 O L (全 6 頁)

(21) 出願番号 特願平9-10821

(22) 出願日 平成9年(1997)1月24日

(71) 出願人 000004237

日本電気株式会社
東京都港区芝五丁目7番1号

(71) 出願人 593162453

技術研究組合新情報処理開発機構
東京都千代田区東神田2-5-12 龍角散
ビル8階

(72) 発明者 山内 宗

東京都港区芝五丁目7番1号 日本電気株
式会社内

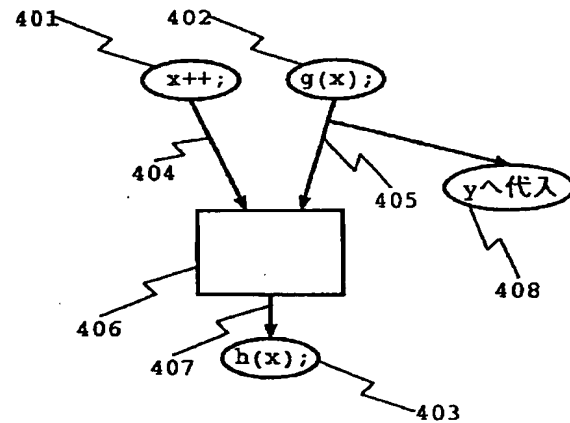
(74) 代理人 弁理士 後藤 洋介 (外2名)

(54) 【発明の名称】 演算装置合成方法

(57) 【要約】

【課題】 導体集積回路にデータ/制御流に従ってプログラムを直接マップして実行する方法において、手続型言語を用いつつデータフローの概念が有する高い並列性を取り出す。

【解決手段】 大域変数と関数の参照代入関係を再帰的に解析するとともに、大域変数と関数呼び出し及び演算の代入関係を保持するスコープスタックを生成する。関数呼び出しを半導体集積回路にマップする際に、その関数が参照している大域変数に対して代入している演算及び関数呼び出しをスコープスタックを逆に辿ってみつけたし、その関数の引数が揃っていること、及びスコープスタックに従い、その関数が参照している大域変数へ代入している演算及び関数呼び出しが終了していること、の2条件が成立したとき関数呼び出しを起動する回路を生成する。



【特許請求の範囲】

【請求項1】 手続型言語により記述されたプログラムを半導体集積回路にマップする演算装置合成方法において、

前記手続型言語により記述された関数、及び参照及び代入のいずれか又は両方を行う大域変数の参照代入関係を再帰的に解析する大域変数解析段階と、

大域変数に対してどの演算及び関数呼び出しが代入するかについての情報を保持するスコープスタックを生成するスコープスタック生成段階と、

前記スコープスタックを成長方向の逆方向に辿り、関数呼び出し甲が参照する大域変数に対して代入する関数呼び出し乙及び演算を前記参照代入関係に基づいて見つけるスコープ解析段階と、

前記関数呼び出し甲の引数が揃い、かつ、前記関数呼び出し乙及び前記演算が終了したとき、関数呼び出し甲を起動する回路を生成する回路生成段階とを含むことを特徴とする演算装置合成方法。

【請求項2】 請求項1記載の演算装置合成方法において、前記回路生成段階は、前記関数呼び出し甲の引数が揃い、かつ、前記関数呼び出し乙における大域変数への代入が終了したとき、関数呼び出し甲を起動する回路を生成することを特徴とする演算装置合成方法。

【請求項3】 請求項1及び2のいずれかに記載の演算装置合成方法において、前記関数呼び出し乙により代入される複数の大域変数が存在する場合、前記回路生成段階は、前記関数呼び出し甲の引数が揃い、かつ、前記関数呼び出し乙における大域変数への代入のうち最も遅く終了する代入が終了したとき、関数呼び出し甲を起動する回路を生成することを特徴とする演算装置合成方法。

【請求項4】 請求項1乃至3のいずれかに記載の演算装置合成方法において、前記関数呼び出し乙により代入される複数の大域変数が存在する場合、前記回路生成段階は、前記関数呼び出し甲の引数が揃い、かつ、前記関数呼び出し乙における大域変数への代入がすべて終了したとき、関数呼び出し甲を起動する回路を生成することを特徴とする演算装置合成方法。

【請求項5】 請求項1乃至4のいずれかに記載の演算装置合成方法において、前記半導体集積回路はPLD (Programmable Logic Device) であることを特徴とする演算装置合成方法。

【請求項6】 請求項1乃至4のいずれかに記載の演算装置合成方法において、前記半導体集積回路は再構成可能であることを特徴とする演算装置合成方法。

【請求項7】 請求項6に記載の演算装置合成方法において、前記再構成可能な半導体集積回路はFPGA (Field Programmable Gate Array) であることを特徴とする演算装置合成方法。

【請求項8】 請求項1乃至7のいずれかに記載の演算装置合成方法において、前記手続型言語はFORTRAN

N及びC言語のいずれかであることを特徴とする演算装置合成方法。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明は半導体集積回路に関し、特に半導体集積回路へのマッピングに関する。

【0002】

【従来の技術】従来、高度な並列処理を目指すアーキテクチャとして、FPGA (Field Programmable Gate Array) などのプログラマブルな半導体集積回路上にデータ/制御流に従ってプログラムを直接マップして実行するという方式がある。プログラムを半導体集積回路にマップする際に、データフローの概念を取り入れると大規模な並列性を取り出すことが容易になる。

【0003】この方式において、プログラムを如何に半導体集積回路にマップするかは重要な技術課題である。単純にデータフロー型計算モデルの概念を取り入れると大規模な並列性を取り出すことは容易になるが、プログラミングモデルとしては使いにくいものとなる。データフロー型計算モデルの概念と相性のよいプログラミングモデルに基づいたプログラム言語として良く知られているのは、単一代入の関数型言語である。

【0004】しかし、世間に広く受け入れられているC言語やFORTRAN等の手続型言語と比べると、単一代入の関数型言語は制約が厳しい言語で、あまり一般的ではなく、使いやすいとは言いがたい。

【0005】逆に、半導体集積回路上にデータ/制御流に従ってプログラムを直接マップする際の記述言語として従来のC言語やFORTRAN等の手続型言語を用いることにも問題が多い。局所変数に関してはスコープに従ってデータ流を解析することによりマップすることが可能であるが、手続型言語につきものの大域変数の存在が厄介である。

【0006】大域変数はプログラムの全領域で参照/代入されるので、データ流と制御流に複雑に影響され、スコープの解析は容易ではない。そして、関数を抜けたらその存在を失う局所変数と異なり、大域変数は常に変数の実体が存在し続けなければならないので、データフロー型計算モデルのようにデータが常に一か所に止まらずに流れ続けることを基本とした計算モデルとは相性が悪い。

【0007】

【発明が解決しようとする課題】本発明が解決しようとする課題は、半導体集積回路上にデータ/制御流に従ってプログラムを直接マップする際の記述言語として手続型言語を用いつつ、大規模な並列性を取り出すことができる演算装置合成方法を提供することである。

【0008】

【課題を解決するための手段】上記のような課題を解決

するため、本発明は、手続型言語における関数起動の呼び出し条件として、データフローの概念に基づいた、○引数の値が揃っていること、という条件の他に、その関数が参照する大域変数に対して、○代入している演算が終了していること、○副作用を与える関数が終了していること、という条件を加えることにより、データフローの概念を拡張し、大域変数という概念を用いる手続型言語でもデータフローの概念が有する高い並列性を取り出すことが可能になるというものである。

【0009】本発明は、手続型言語により記述されたプログラムを半導体集積回路にマップする演算装置合成方法において、前記手続型言語により記述された関数、及び参照及び代入のいずれか又は両方を行う大域変数の参照代入関係を再帰的に解析する大域変数解析段階と、大域変数に対してどの演算及び関数呼び出しが代入するかについての情報を保持するスコープスタックを生成するスコープスタック生成段階と、前記スコープスタックを成長方向の逆方向に辿り、関数呼び出し甲が参照する大域変数に対して代入する関数呼び出し乙及び演算を前記参照代入関係に基づいて見つけるスコープ解析段階と、前記関数呼び出し甲の引数が揃い、かつ、前記関数呼び出し乙及び前記演算が終了したとき、関数呼び出し甲を起動する回路を生成する回路生成段階とを含むことを特徴とする演算装置合成方法を提供する。

【0010】これにより、世間で広く用いられている手続型言語をデータフロー的な概念に基づきマップすることが可能になり、使いやすさと高性能を両立することが可能になる。

【0011】

【発明の実施の形態】本発明の第1の実施の形態について図面を参照して説明する。

【0012】図1に入力プログラムの例を示す。このプログラムを半導体集積回路に直接マップして実現することを考え、特に関数呼び出しh(x)103をどのタイミングで起動するかについて着目する。大域変数a109が存在しなければ、関数呼び出しh(x)の引数x104の値がそろった時点で関数呼び出しh(x)103を起動することが可能である。しかし、大域変数が存在する場合には、そのデータ依存関係も考慮する必要がある。

【0013】そこで、まず、各関数呼び出しで呼び出されるサブルーチンにおいて、どの大域変数が参照され、どの大域変数には代入がなされるのかを解析する。解析は図2に示すように、関数f(x)の大域変数参照表201及び大域変数代入表202、関数g(x)の大域変数参照表203及び大域変数代入表204、並びに関数h(x)の大域変数参照表205及び大域変数代入表206を生成して行う。

【0014】ここで注意しなければならないのは、関数f(x)と大域変数のデータ依存関係は、一見すると大域

変数bへの代入106しか存在しないかのように見えるが、関数f(x)から呼ばれる関数g(x)及び関数h(x)において大域変数aへの代入が行われていることである。結果として、関数f(x)は大域変数bだけではなく、大域変数aともデータ依存関係があることになる。このように、ある関数自身の中で直接のデータ依存関係はないが、その関数の中で呼び出される他の関数の中でデータ依存関係があるような大域変数については、関数の中で呼ばれる関数がどの大域変数に対して代入するかの一覧を再帰的に実行することにより解析を行う。

【0015】次に、どの大域変数に対してどこで代入が行われているかを示すスコープスタックを生成する。図3は図1の入力プログラム及び図2に示した表に従って生成されたスコープスタック301を示した図である。この場合は関数呼び出しf(x)におけるスコープスタックであり、プログラムを記述の順番に解析するのに連れてスコープスタック301は図3の右方向に成長する。そして、スコープスタック301の各項目には変数名、局所変数及び大域変数の区別と、その場での代入なのか、関数呼び出しによる大域変数への代入なのかの区別とを記し、その場での代入ならば×印を、関数呼び出しによる大域変数への代入の場合にはその関数名を記す。

【0016】ここで言う関数呼び出しによる大域変数への代入とは、その関数を呼び出すことにより、呼び出された関数の中で大域変数に対する代入が行われるということであり、どの関数呼び出しでどのような代入が行われるかは図2を参照すればわかる。

【0017】スコープスタック301を用いると、プログラムのある時点で変数への参照が生じた場合に、参照すべき値はどの代入の結果であるかを知ることができる。即ち、ある変数への参照が生じたら、その時点のスコープスタック301を逆向きに辿り、最初に見つけたその変数への代入が参照すべき値となるのである。

【0018】そしていよいよ関数呼び出しh(x)103をどのタイミングで起動するかを決定する。関数呼び出しh(x)103を起動するのに必要なのは、引数xの値が求められている事、及び関数呼び出しh(x)103が参照する大域変数aの値が求められている事である。そこで、スコープスタック301を逆向きに辿って引数x及び大域変数aがそれぞれどの時点で求められるのかを判定する。すると、引数xについては引数xへの参照及び代入105で代入されたときの値を待つ必要があり、大域変数aについては関数呼び出しg(x)101で代入されるので関数呼び出しg(x)101の終了を待つ必要があることがわかる。

【0019】従って、関数呼び出しh(x)103を起動するには、引数xへの参照と代入105、及び関数呼び出しg(x)101の双方の終了を待てばよい。図4は関数呼び出しh(x)を起動する信号を生成する例を示す図

である。図4を参照すると、「x++」の処理をするブロック401が処理を終了したことを示す終了信号404、及び関数g(x)を呼び出すブロック402が終了したことを示す終了信号405の双方の信号が揃ったことを各終了信号の同期をとる同期化ブロック406によって検出されると、同期化ブロック406は関数h(x)の起動信号407を生成する。

【0020】以上のようにすることにより、大域変数による依存関係が生じる従来のC言語やFORTRANなどの手続型言語をデータフロー的な概念に基づきマップすることが可能になり、使いやすさと高度な並列性を両立することが可能になる。

【0021】図5は本発明の第2の実施の形態による関数h(x)503の起動を示した図である。図1のプログラムを見ると、関数呼び出しh()103を起動するには必ずしも関数呼び出しg()101の実行が終了するのを待つ必要がないことに気付く。関数呼び出しh()103が必要としているのは大域変数aの値なので、関数呼び出しg()101の中の大域変数aへの参照と代入107が終わった時点ですぐに関数呼び出しh()103を起動しても構わないのである。この点を考慮したのが第2の実施の形態である。

【0022】図4では、「x++」の処理の終了信号404及び関数g(x)の終了信号405の双方の信号が揃ったことを各終了信号の同期化ブロック406により検出していたが、図5では、「x++」の処理の終了信号504、及び大域変数aへの参照及び代入107の直後に関数g(x)における大域変数aへの代入の終了信号509の双方を各終了信号の同期化ブロック506によって検出し、双方が揃うと同期化ブロック506は関数h(x)の起動信号507を生成する。これにより、関数呼び出しg()101の終了を待たずして関数呼び出しh()103を起動することが可能となり、実行時間の短縮を図ることができる。

【0023】図6は本発明の第3の実施の形態の対象となるプログラムの例である。第1及び第2の実施の形態では、大域変数aについてのみ待ち合わせをすればよかった。これに対し、複数の大域変数について依存関係がある場合について図6を参照して説明する。

【0024】関数呼び出しg()601において大域変数a、b、cに代入がされ、関数呼び出しh()602において大域変数a、b、cへの参照が生じる。関数呼び出しg()601における大域変数a、b、cへの代入603、604、605の順番の依存関係が明確であり、大域変数cへの代入605が最後に実行される。従って、それらの複数の大域変数への代入のうち、最後の代入終了信号、即ち大域変数cへの代入605の代入終了信号を代表の代入終了信号として用いることが可能である。

【0025】図7は本発明の第4の実施の形態の対象と

なるプログラムの例である。第4の実施の形態も第3の実施の形態と同様に複数の大域変数を扱うが、第3の実施の形態では大域変数どうしの順番の依存関係が明確なプログラムを対象にしたのに対し、第4の実施の形態では順番が特定できないようなプログラムを対象とする。

【0026】大域変数a、b、cへの代入703、704、705はそれぞれ関数呼び出しの結果を代入しているので、代入が終了する順番を特定できない。このため、最後に代入が終了する大域変数を特定できず、第3の実施の形態のように代表の代入終了信号を決めることができない。これに対して、第4の実施の形態では大域変数aへの代入703の代入終了信号、大域変数bへの代入704の代入終了信号、及び大域変数cへの代入705の代入終了信号の3つの代入終了信号が全て揃ったことを検出する同期化ブロックを追加し、その出力を関数呼び出しg()701の代入終了信号として用いる。これにより、複数の大域変数への代入の終了する順番を特定できないプログラムに対しても対応することができる。

【0027】

【発明の効果】本発明によれば、手続型言語における関数起動の呼び出し条件として、データフローの概念に基づいた、○引数の値が揃っていること、という条件の他に、その関数が参照する大域変数に対して、○代入している演算が終了していること、○副作用を与える関数が終了していること、という条件を加えることにより、データフローの概念を拡張し、大域変数という概念を用いる手続型言語でもデータフローの概念が有する高い並列性を取り出すことが可能になるというものである。

【0028】本発明による演算装置合成方法は、各関数が参照/代入する大域変数について再帰的に解析する段階と、ある大域変数に対してどの演算や関数呼び出しが代入しているかという情報を保持するスコープスタックを生成する段階と、ある関数呼び出しを半導体集積回路にマップする際に、その関数が参照している大域変数に対して代入している演算や関数呼び出しを、スコープスタックを逆に辿って見つけた段階と、ある関数の呼び出し条件として、○その関数の引数が揃っていること、○スコープスタックに従い、その関数が参照している大域変数へ代入している演算及び関数呼び出しが終了していること、の2条件を集め、それらの全ての条件が揃った時点で関数呼び出しを起動する回路を生成する段階で構成される。

【0029】これにより、世間で広く用いられている手続型言語をデータフロー的な概念に基づきマップすることが可能になり、使いやすさと高性能を両立することが可能になる。

【0030】以上、本発明を実施の形態に基づいて説明したが、本発明はこれに限定されるものではなく、当業者の通常の知識の範囲内でその変更や改良が可能である

ことは勿論である。

【図面の簡単な説明】

【図1】本発明の第1の実施の形態の対象となるプログラムの例である。

【図2】図1のプログラムの大域変数参照表及び大域変数代入表である。

【図3】図1のプログラム及び図2に示した表に従って生成されたスコープスタックを示した図である。

【図4】関数呼び出しh()を起動する信号を生成する例を示す図である。

*【図5】本発明の第2の実施の形態による関数h(x)の起動を示した図である。

【図6】本発明の第3の実施の形態の対象となるプログラムの例である。

【図7】本発明の第4の実施の形態の対象となるプログラムの例である。

【符号の説明】

201、203、205 大域変数参照表

202、204、206 大域変数代入表

*10 301 スコープスタック

【図1】

```

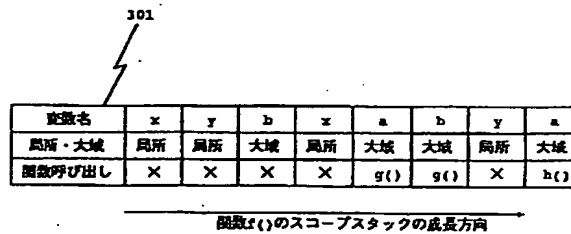
/* global variable */
int a = 0;
int b = 0;

void
f()
{
    int x = 0;
    int y = 0;
    b = 3;
    x++;
    y = g(x);
    h(x);
}

int
g(p)
int p;
{
    a++;
    p += b++;
    return p;
}

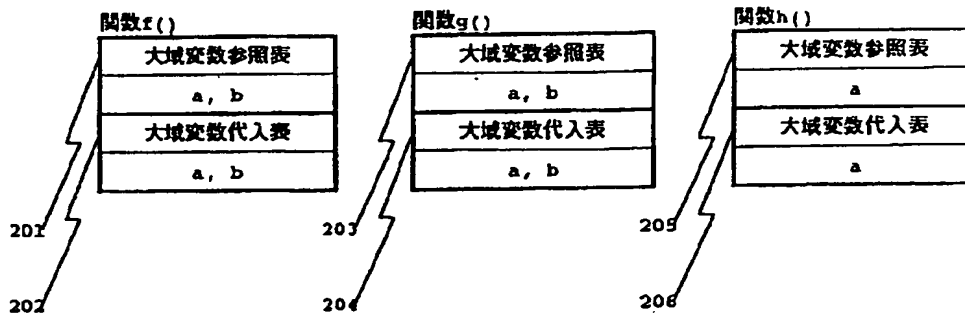
void
h(p)
int p;
{
    a = a + p;
}

```

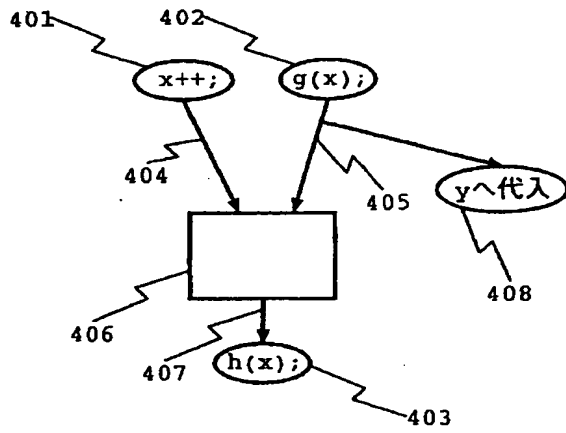


【図3】

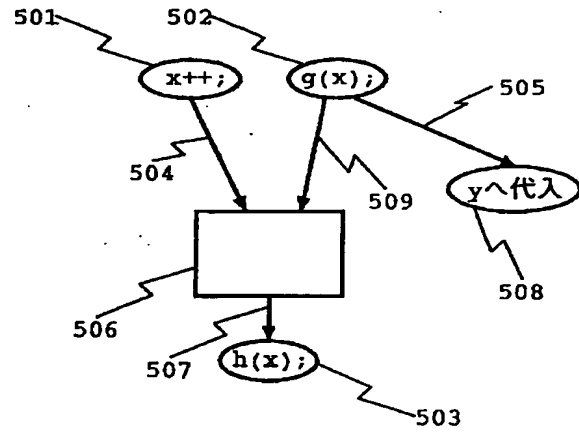
【図2】



【図4】



【図5】



【図6】

```

/* global variable */
int a = 0;
int b = 0;
int c = 0;

void
f()
{
  g();
  h();
}

void
g()
{
  a = 1;
  b = a + 2;
  c = a + b;
}

void
h()
{
  a = a + b + c;
}

```

【図7】

```

/* global variable */
int a = 0;
int b = 0;
int c = 0;

void
f()
{
  g();
  h();
}

void
g()
{
  a = p();
  b = q();
  c = r();
}

void
h()
{
  a = a + b + c;
}

```